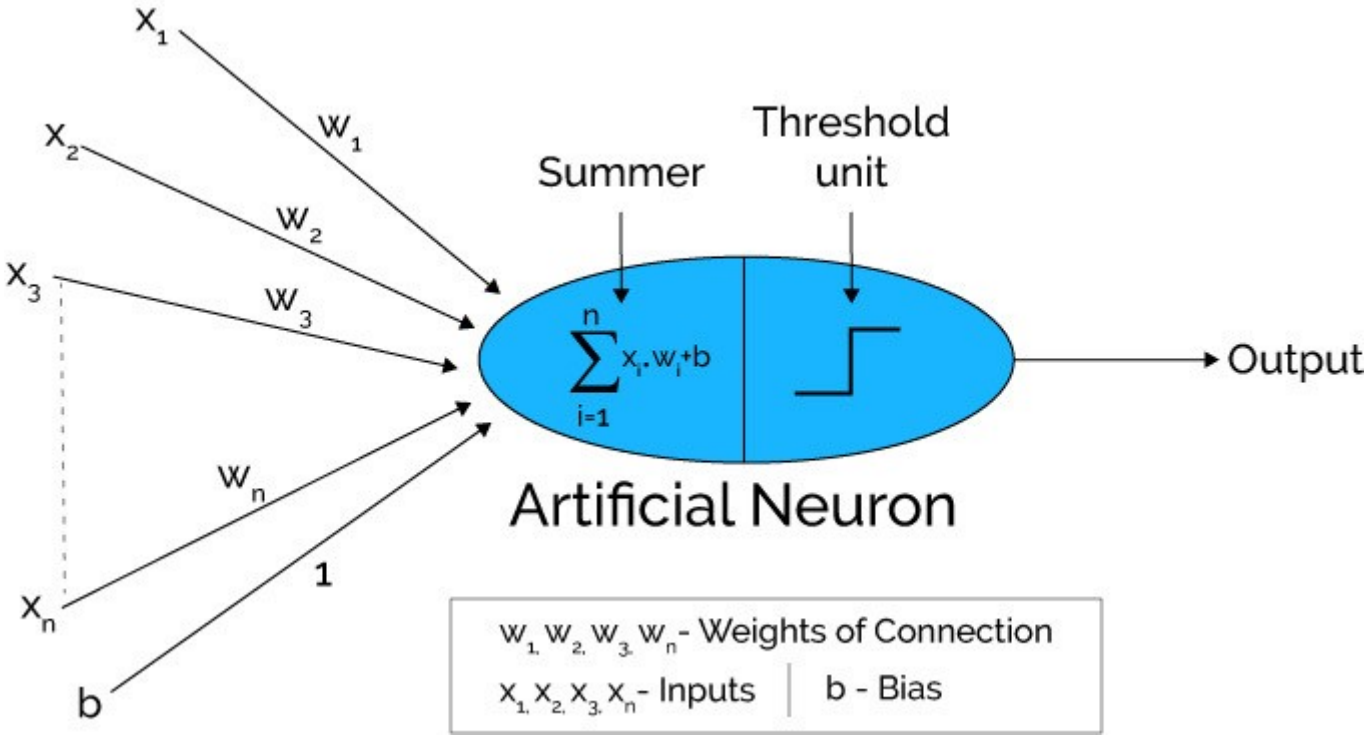
姓名：徐屹恒

时间：2020.1--2020.2

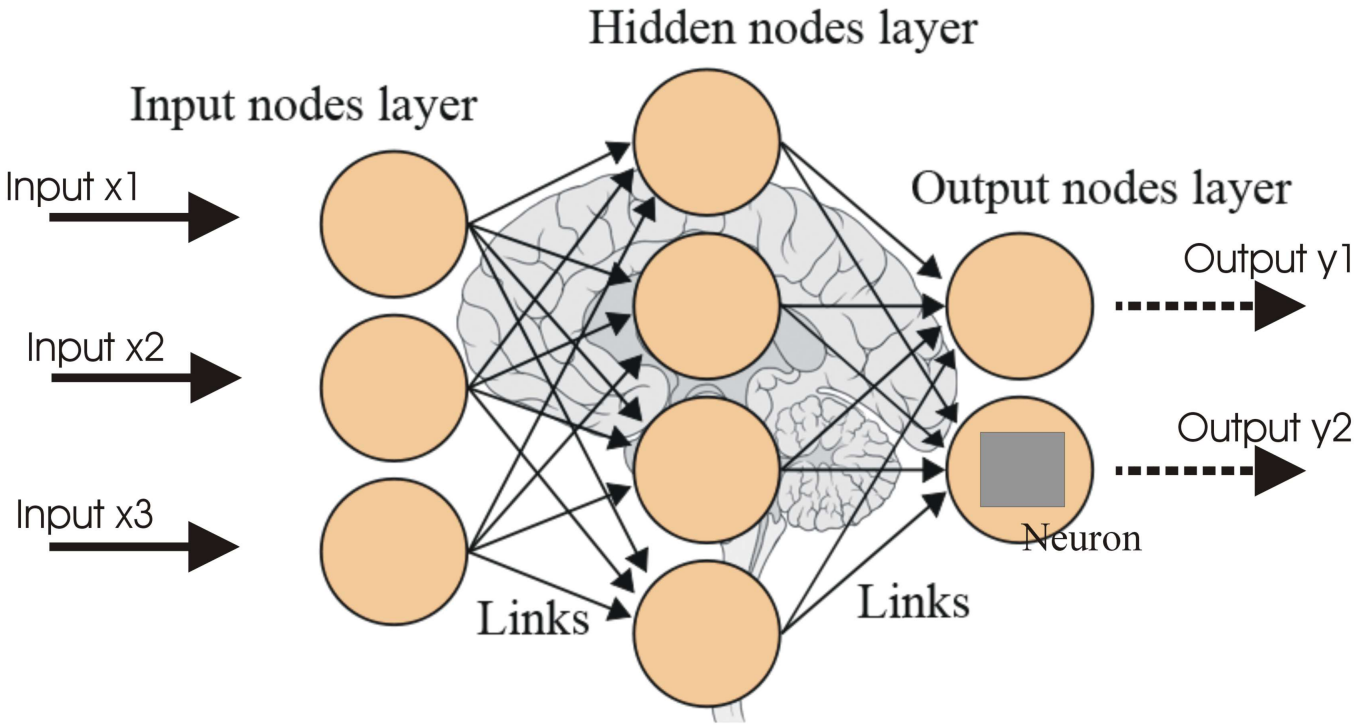
题目：BP神经网络模拟XOR函数

有一个单细胞生物，生活在糖水和盐水混合物中，糖水盐水在不停的流动，它有两个鞭毛，分别可以感知糖水和盐水，当总浓度到达一定值时，就会吃这些糖水盐水



但是，同时吃进糖水和盐水，会造成细胞结石，有损细胞健康

慢慢地，单细胞生物慢慢和其它细胞一起，组成了多细胞生物，进化出了一个能力，有糖水吃糖水，有盐水吃盐水，同时都有的话就不吃，防止细胞里面无法同时消化两种物体。



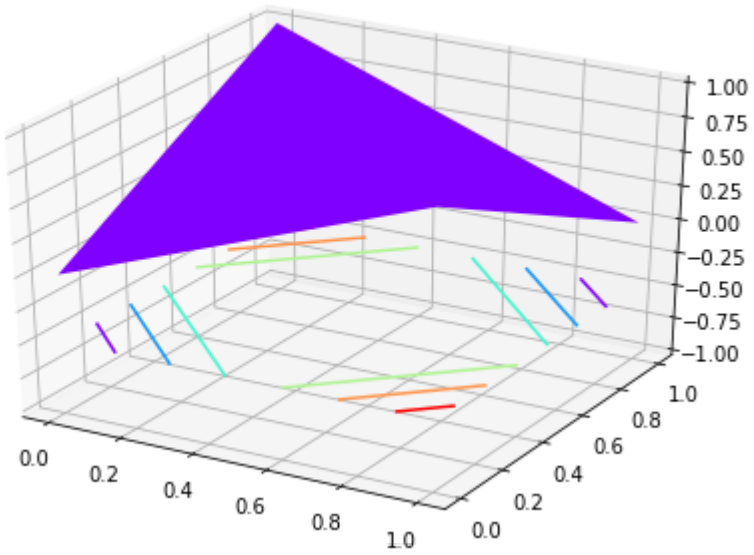
下面，我们用人工神经元网络来模拟从单细胞到多细胞的进化过程

首先，画出最终的输入输出函数3维曲线。

目标输出曲线

In [35]:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure()
ax = Axes3D(fig)

x1=np.arange(0,2,1)
x2=np.arange(0,2,1)

X,Y=np.meshgrid(x1, x2)     # x-y 平面的网格
Z=np.logical_xor(X,Y)

ax.plot_surface(X, Y, Z, rstride = 1, cstride = 1, cmap = plt.get_cmap('rainbow'))
ax.contour(X, Y, Z, offset = -0.5, cmap = 'rainbow')
ax.set_zlim(-1, 1)

plt.show()

print(X)
print(Y)
print(Z)
```




```
[[0 1]
 [0 1]]
[[0 0]
 [1 1]]
[[False  True]
 [ True False]]
```

用神经网络来学习XOR函数迭代版

首先，定义一个sigmod函数

In [4]:

```
def nonlin(x, deriv=False):
    if(deriv==True):
        return (x*(1-x))

    return 1/(1+np.exp(-x))
```

1/(1+exp(-x)) 导数推导

(1/(1+exp(-x)))'=((1+exp(-x))^-1)'=(-1)((1+exp(-x))^-2)(1+exp(-x))'=(-1)((1+exp(-x))^-2)(exp(-x))' 而(exp(-x))'可以先转成(exp(x)^-1)'，于是她又是一个复合函数的求导，即(exp(x)^-1)对exp(x)的导数再乘上exp(x)对x的导数，又基本初等函数求导公式告诉我们，(exp(x))'=exp(x)，所以(exp(-x))'=(exp(x)^-1)'=(-1)(exp(x)^-2)(exp(x))'=(-1)(exp(x)^-2)exp(x)=(-1)(exp(x)^-1)=(-1)exp(-x) 那么： (-1)((1+exp(-x))^-2)(exp(-x))'=(-1)((1+exp(-x))^-2)(-1)exp(-x)=exp(-x)((1+exp(-x))^-2)=exp(-x)/((1+exp(-x))^2)

$$f(net) = \frac{1}{1+e^{-net}}$$

$$f'(net) = \frac{e^{-net}}{(1+e^{-net})^2}$$

$$= \frac{1+e^{-net}-1}{(1+e^{-net})^2}$$

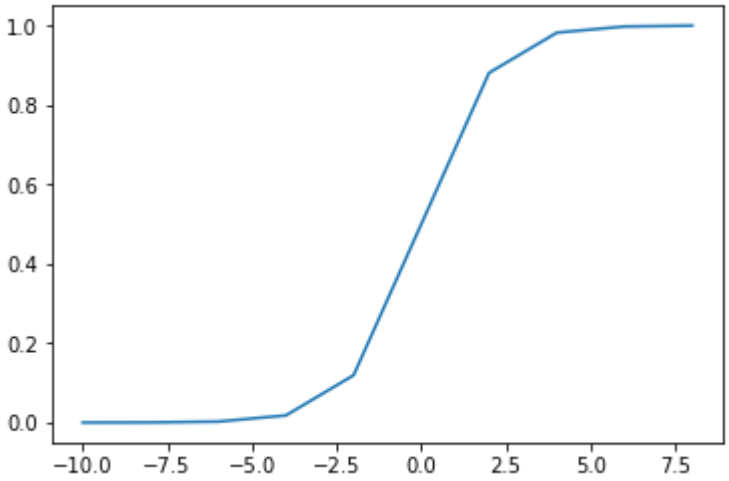
$$= \frac{1}{1+e^{-net}} - \frac{1}{(1+e^{-net})^2}$$

$$= y(1-y)$$

In [5]:

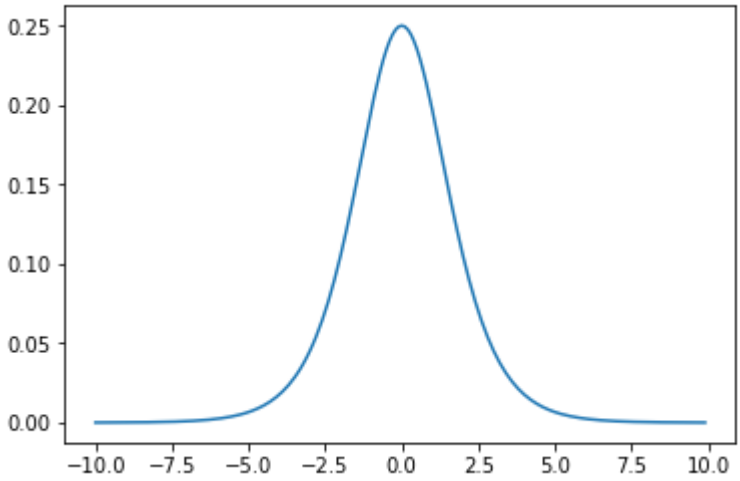
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

x=np.arange(-10,10,2)
y=nonlin(x)
print(y)
plt.plot(x,y)
plt.show()
```

```
[4.53978687e-05 3.35350130e-04 2.47262316e-03 1.79862100e-02
 1.19202922e-01 5.00000000e-01 8.80797078e-01 9.82013790e-01
 9.97527377e-01 9.99664650e-01]
```



In [6]:

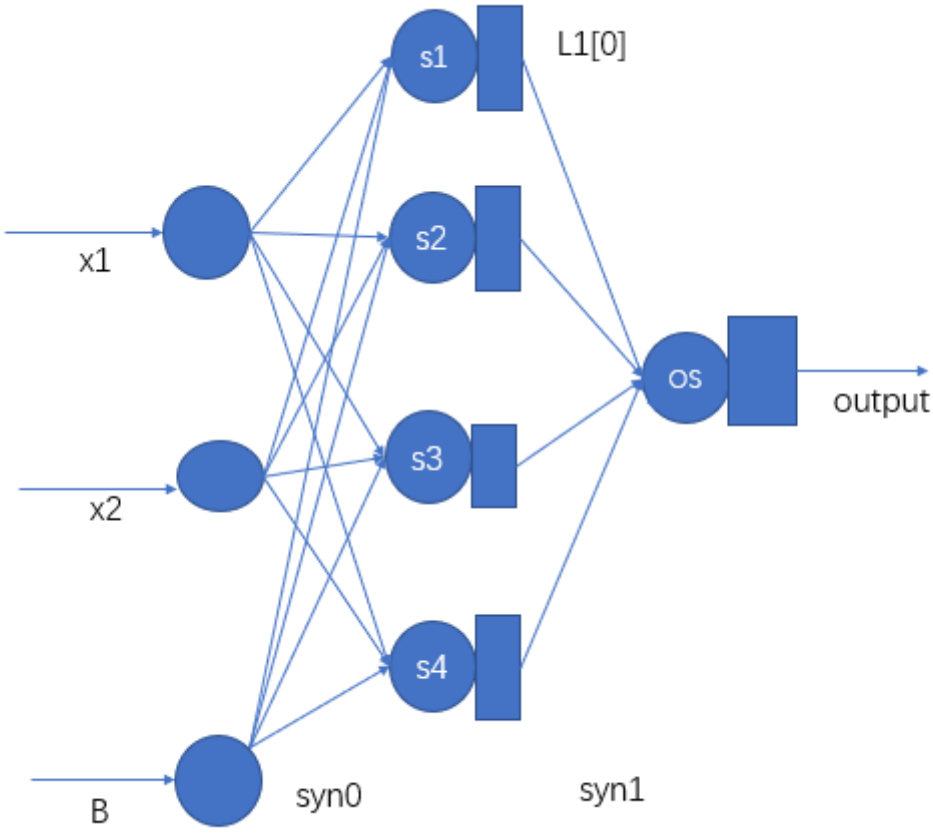
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

x=np.arange(-10,10,0.1)
y=1/(1+np.exp(-x))
yderiv=nonlin(y,deriv=True)
plt.plot(x,yderiv)
plt.show()
```



表格试凑法

B	x1	x2	WB1	WB2	WB3	WB4	W11	W12	W13	W14	W21	W22	W23	W24	S1	S2	S3	S4	TS1	TS2	TS3	TS4	L2w1	L2W2	L2W3	L2W4	output
1	0	0	10	0.2	-1	-20	-10	-20	20	20	-10	20	-20	20	10	0.2	-1	-20	1	0	0	0	0	1	1	0	0
1	0	1	10	0.2	-1	-20	-10	-20	20	20	-10	20	-20	20	0	20	-21	0	0	1	0	0	0	1	1	0	1
1	1	0	10	0.2	-1	-20	-10	-20	20	20	-10	20	-20	20	0	-20	19	0	0	0	1	0	0	1	1	0	1
1	1	1	10	0.2	-1	-20	-10	-20	20	20	-10	20	-20	20	-10	0.2	-1	20	0	0	0	1	0	1	1	0	0

	WB1	WB2	WB3	WB4
WB1	10	0.2	-1	-20
W11	-10	-20	20	20
W21	-10	20	-20	20
	W21	W22	W23	W24



In [47]:

```
syn0=np.array([[-20.0,-20.0,20.0,20.0],
               [-20.0,20.0,-20.0,20.0],
               [20.0,0.0,0.0,-20.0]])
#print(syn0)

syn1=np.array([[0],
               [1.0],
               [1.0],
               [0]])
```

所有可能的输入值放在一个向量中

In [8]:

```
#input data
xinputs = np.array([[0,0,1],
            [0,1,1],
            [1,0,1],
            [1,1,1]])
l0=xinputs
```

计算第一层的线性输出

In [42]:

```
S01=np.dot(xinputs,syn0)
#print(S01)
```

In [43]:

```
l1=nonlin(S01)
#print(l1)
```

In [44]:

```
S02=np.dot(l1,syn1)
#print(S02)
```

In [46]:

```
l2=nonlin(S02)
#print(l2)
```

希望的的输出结果

In [13]:

```
#output data
y = np.array([[0],
              [1],
              [1],
              [0]])
```

误差是

In [14]:

```
l2_error=y-l2
print(l2_error)
```

```
[[-0.73105858]
 [ 0.26894142]
 [ 0.26894142]
 [-0.73105858]]
```

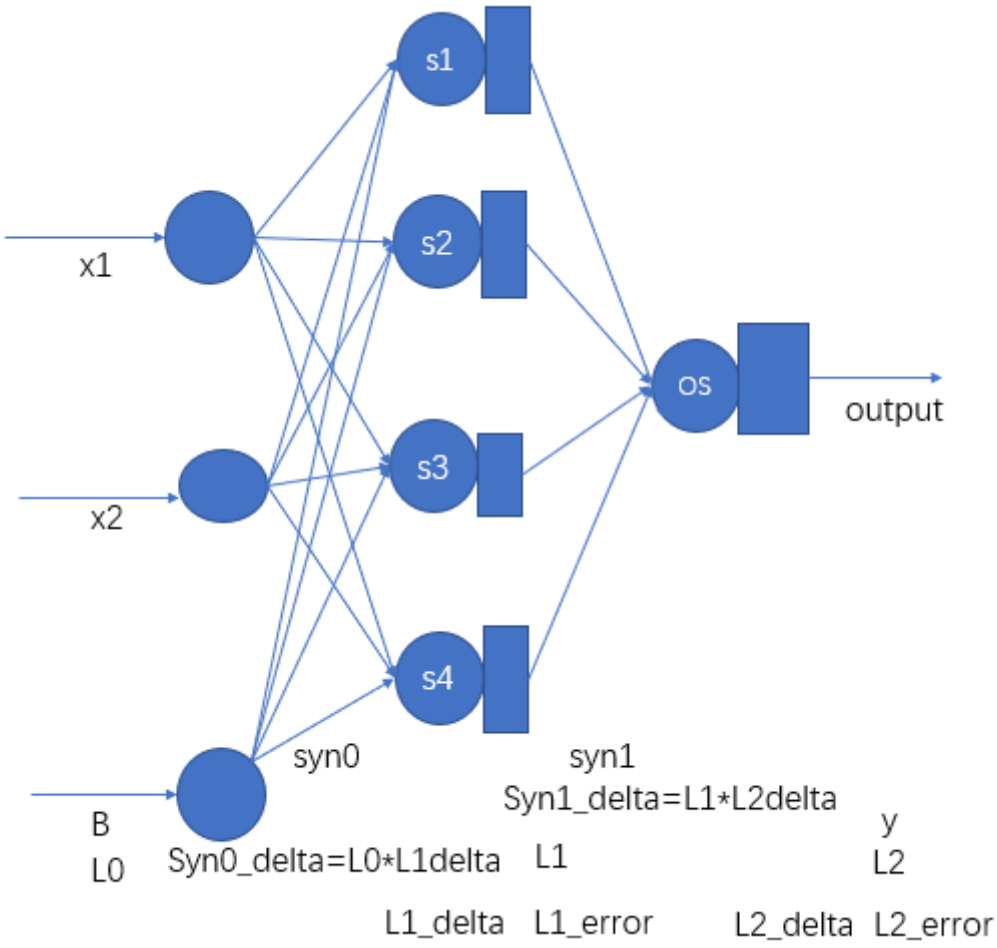
为了直观地得出总误差多大，引入绝对值平均值lmabs

In [15]:

```
#误差的平均绝对为
lms=np.mean(np.abs(l2_error))
print(lms)

sk=[]
yerrorlmss=[]
syn1_0=[]
sks=0

syn1_0.append(syn1[0,0])
sk.append(sks)
yerrorlmss.append(lms)
sks=sks+1
```

0.5

误差反向传播



利用误差修正第二层权值网络

In [16]:

```
l2_delta = l2_error*nonlin(l2, deriv=True)
```

In [17]:

```
print(l2_delta)
```

```
[[-0.14373484]
 [ 0.05287709]
 [ 0.05287709]
 [-0.14373484]]
```

计算第一层的误差

In [41]:

```
#print(syn1)
```

In [39]:

```
#print(syn1.T)
```

In [40]:

```
l1_error = l2_delta.dot(syn1.T)
print(l1_error)
```

```
[[ 0.02819802 -0.02700121 -0.02700121  0.02334109]
 [-0.02527127  0.02419868  0.02419868 -0.02091845]
 [-0.02527127  0.02419868  0.02419868 -0.02091845]
 [ 0.02805741 -0.02686657 -0.02686657  0.0232247 ]]
```

计算第一层的修正值

In [37]:

```
l1_delta = l1_error * nonlin(l1,deriv=True)
#print(l1_delta)
```

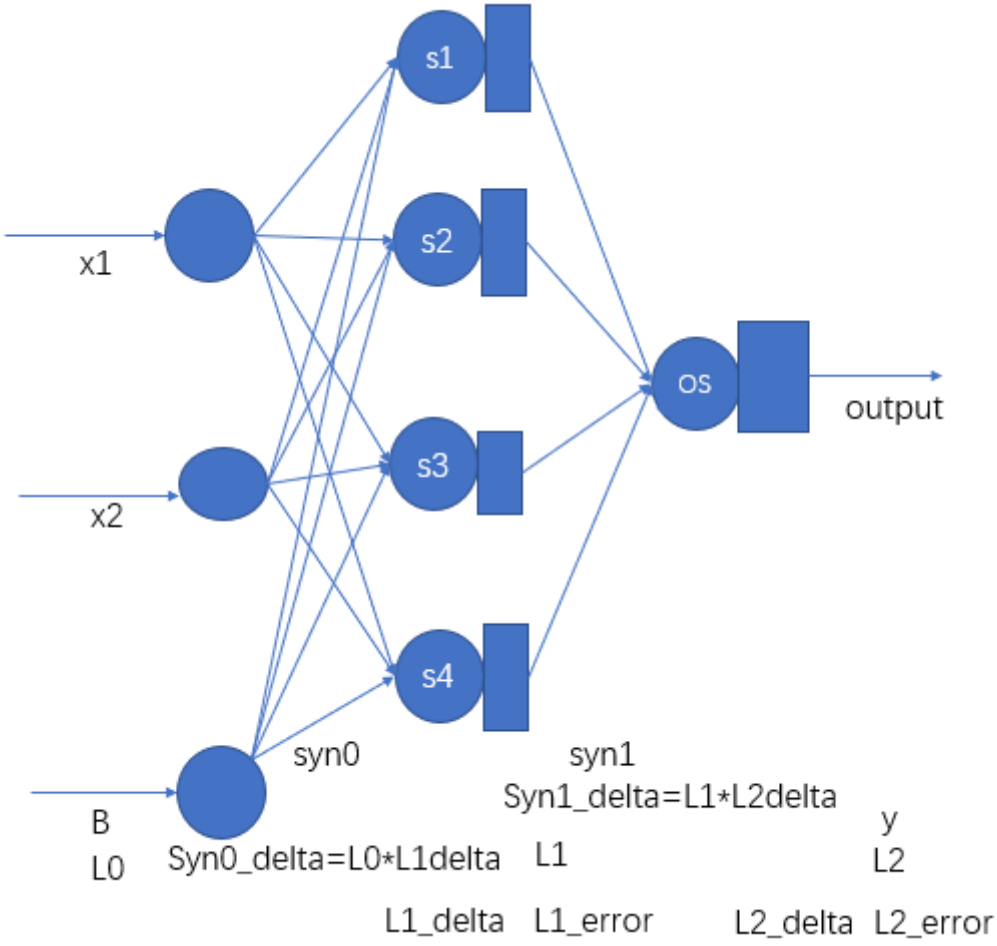
修正权值网络

In [22]:

```
syn1 += l1.T.dot(l2_delta)
syn0 += l0.T.dot(l1_delta)
```

In [36]:

```
#print(syn0)
#print(syn1)
```


s1
s2
s3
s4
x1
x2
B
L0
os
output
syn0
Syn0_delta=L0*L1delta
syn1
Syn1_delta=L1*L2delta
L1
y
L2
L1_delta
L1_error
L2_delta
L2_error

机器迭代结果

In [24]:

```
for j in range(100):

    print("start forward calulate")
    l0 = xinputs
    print("l0")
    print(l0)
    l1 = nonlin(np.dot(l0, syn0))
    print("l1")
    print(l1)
    l2 = nonlin(np.dot(l1, syn1))
    print("l2")
    print(l2)

    # Back propagation of errors using the chain rule.
    l2_error = y - l2

    print("l2")
    print(l2)

    print("l2_error")
    print(l2_error)

    lms=np.mean(np.abs(l2_error))
    syn1_0.append(syn1[0,0])
    sk.append(sks)
    yerrorlmss.append(lms)
    sks=sks+1

    print("Error:")
    print(lms)

    l2_delta = l2_error*nonlin(l2, deriv=True)

    print("nonline derive")
    print(nonlin(l2,deriv=True))

    print("l2_delta")
    print(l2_delta)

    l1_error = l2_delta.dot(syn1.T)

    print("syn1.T")
    print(syn1.T)

    print("l1_error")
    print(l1_error)

    l1_delta = l1_error * nonlin(l1,deriv=True)

    print("nonlin(l1,deriv=True)")
    print(nonlin(l1,deriv=True))


    #update weights (no learning rate term)
    syn1 += l1.T.dot(l2_delta)
    syn0 += l0.T.dot(l1_delta)

    print("syn0 syn1")
```

```
    print(syn0)

    print(syn1)

print("Output after training")
print(l2)

print(syn1_0)
print(yerrorlmss)
```

```
nonlin(l1,deriv=True)
[[9.74579197e-09 1.10834492e-01 1.10834492e-01 5.07543896e-10]
 [8.07829551e-02 2.84670782e-08 1.49237433e-10 9.70309061e-02]
 [8.07829551e-02 1.49237433e-10 2.84670782e-08 9.70309061e-02]
 [9.21927516e-11 3.35871724e-02 3.35871724e-02 3.39925554e-08]]
syn0 syn1
[[-20.77896885 -20.69826051  19.30173956  19.29711696]
 [-20.77896885  19.30173956 -20.69826051  19.29711696]
 [ 18.44206233  -1.93237857  -1.93237857 -21.40576615]]
[[-2.75425392]
 [ 2.6419449 ]
 [ 2.6419449 ]
 [-2.27336881]]
start forward calulate
10
[[0 0 1]
 [0 1 1]
 [1 0 1]
 [1 1 1]]
11
```

迭代过程中误差的变化曲线